Hibernate Queries and Annotations

Hibernate Introduction –

Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.

Session and Session Factory –

SessionFactory In Hibernate

SessionFactory is an interface available in org.hibernate package which extends Referenceable and Serializable interface and provides factory methods to get session object. These are some key points about SessionFactory –

- SessionFactory is thread safe so multiple threads can access the SessionFactory at the same time.
- SessionFactory is Immutable. Once we create SessionFactory we can not modify it(If you look

SessionFactory methods we don’t have any setter kind of method)

- SessionFactory is created at the time of application startup, it reads all information from the configuration file(hibernate.cfg.xml file). We will see later in details.
- We should have one SessionFactory for one database configuration.
- We can create a SessionFactory object using the Configuration class as below.

```
SessionFactory sessionFactory = new
Configuration().configure().buildSessionFactory();
```

Session In Hibernate

Session is an interface available in `org.hibernate` package which provides different API to communicate java application to hibernate. Some key points related to session –

- The session is not thread-safe.
- The main use of the Session object to perform create, get, and delete operations for java classes(entity).
- The first level cache belongs to the session object.
- We can have multiple sessions for a SessionFactory.

- We can get Session object using SessionFactory reference as below.

```
Session session = sessionFactory.openSession();
```

Hibernate Annotations

Hibernate annotations are the newest way to define mappings without the use of XML file. You can use annotations in addition to or as a replacement of XML mapping metadata.

Hibernate Annotations is the powerful way to provide the metadata for the Object and Relational Table mapping. All the metadata is clubbed into the POJO java file along with the code, this helps the user to understand the table structure and POJO simultaneously during the development.

If you going to make your application portable to other EJB 3 compliant ORM applications, you must use annotations to represent the mapping information.

There are many different types of annotations in hibernate, some of them are –

@Entity Annotation

The EJB 3 standard annotations are contained in the **javax.persistence** package, so we import this package as the first step. Second, we used the **@Entity** annotation to the Employee class, which marks this class as an entity bean, so it must have a no-argument constructor that is visible with at least protected scope.

@Table Annotation

The @Table annotation allows you to specify the details of the table that will be used to persist the entity in the database.

The @Table annotation provides four attributes, allowing you to override the name of the table, its catalogue, and its schema, and enforce unique constraints on columns in the table.

@Id and @GeneratedValue Annotations

Each entity bean will have a primary key, which you annotate on the class with the **@Id** annotation. The primary key can be a single field or a combination of multiple fields depending on your table structure.

By default, the @Id annotation will automatically determine the most appropriate primary key generation strategy to be used but you can override this by applying the **@GeneratedValue** annotation, which takes two parameters **strategy** and **generator** that I'm not going to discuss here, so let us use only the default key generation strategy. Letting Hibernate determine which generator type to use makes your code portable between different databases.

@Column Annotation

The @Column annotation is used to specify the details of the column to which a field or property will be mapped. You can use column annotation with the following most commonly used attributes −

- **name** attribute permits the name of the column to be explicitly specified.
- **length** attribute permits the size of the column used to map a value particularly for a String value.
- **nullable** attribute permits the column to be marked NOT NULL when the schema is generated.
- **unique** attribute permits the column to be marked as containing only unique values.

Hibernate Crud operations

A CRUD operation deals with creating, retrieving, updating, and deleting records from the table. In this tutorial we will see how it is done using Hibernate annotations. We are going to discuss 4 main functionalities:

- Creating a Record
- Displaying Records
- Updating a Record
- Deleting a Record

CRUD means the basic operations to be done in a data repository. We directly handle records or data objects; apart from these operations, the records are passive entities. CRUD stands for **C**reate, **R**ead, **U**pdate, and **D**elete. The CRUD functions are the user interfaces to databases, as they permit users to create, view, modify and alter data. CRUD works on entities in databases and manipulates these entities.

For instance, a simple student database table adds (creates) new student details, accesses (reads) existing student details, modifies (updates) existing student data such as subjects, and deletes student details when students leave the school.

The commands corresponding to these operations in SQL are INSERT, SELECT, UPDATE, and DELETE. INSERT adds new records, SELECT retrieves or selects existing records based

on selection conditions, UPDATE modifies existing records and DELETE removes tables or records in a table.

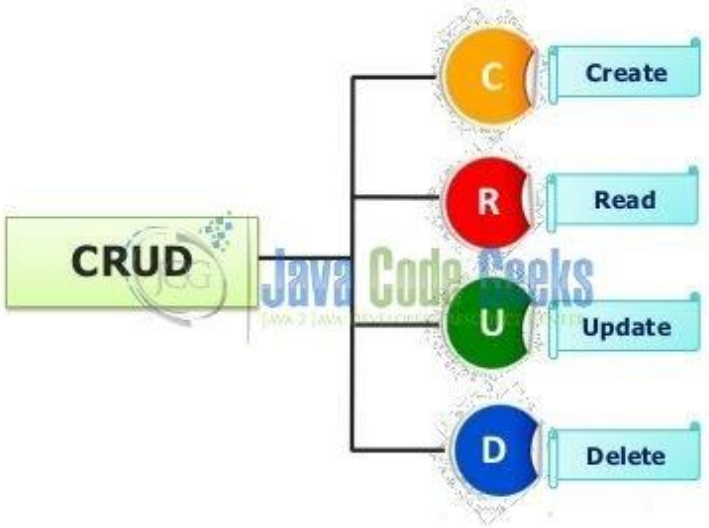

Image Reference: https://www.javacodegeeks.com/

Queries in Hibernate

Hibernate Query Language (HQL) is an object-oriented query language, similar to SQL, but instead of operating on tables and columns, HQL works with persistent objects and their properties. HQL queries are translated by Hibernate into conventional SQL queries, which in turns perform action on database.

Although you can use SQL statements directly with Hibernate using Native SQL, but I would recommend to use HQL whenever possible to avoid database portability hassles, and to take advantage of Hibernate's SQL generation and caching strategies.

Keywords like SELECT, FROM, and WHERE, etc., are not case sensitive, but properties like table and column names are case sensitive in HQL.

There are some clause in Hibernate Query –

FROM Clause

You will use **FROM** clause if you want to load a complete persistent objects into memory.

AS Clause

The **AS** clause can be used to assign aliases to the classes in your HQL queries, especially when you have the long queries.

The **AS** keyword is optional and you can also specify the alias directly after the class name.

SELECT Clause

The **SELECT** clause provides more control over the result set then the from clause. If you want to obtain few properties of objects instead of the complete object, use the SELECT clause.

WHERE Clause

If you want to narrow the specific objects that are returned from storage, you use the WHERE clause.

ORDER BY Clause

To sort your HQL query's results, you will need to use the **ORDER BY** clause. You can order the results by any property on the objects in the result set either ascending (ASC) or descending (DESC).

GROUP BY Clause

This clause lets Hibernate pull information from the database and group it based on a value of an attribute and, typically, use the result to include an aggregate value.

Using Named Parameters

Hibernate supports named parameters in its HQL queries. This makes writing HQL queries that accept input from the user easy and you do not have to defend against SQL injection attacks.

UPDATE Clause

The **UPDATE** clause can be used to update one or more properties of an one or more objects.

DELETE Clause

The **DELETE** clause can be used to delete one or more objects.

INSERT Clause

HQL supports **INSERT INTO** clause only where records can be inserted from one object to another object.